Alma Mater Studiorum · Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

PROGETTAZIONE E REALIZZAZIONE DI UN PORTALE MULTI-ENTE

Tesi di Laurea in Informatica

Relatore:
Chiar.mo Prof.
RENZO DAVOLI

Presentata da:
DIEGO ZUCCATO

Sessione III
A.A. 2017/2018

A tutta la mia famiglia

per avermi s[uo]pportato in questo

(troppo...) lungo percorso.

Introduzione

Quando il Dipartimento di Astronomia (dove lavoravo) si è unito al Dipartimento di Fisica, nell'ormai lontano 2013 mi è stato chiesto di realizzare un sito web per avere l'elenco telefonico del nuovo Dipartimento.

Che "errore" aver accettato! Un applicativo abbastanza banale è cresciuto con 'moduli' per gestire i dati di contatto riservati, tracciare le informazioni relative al parco macchine installato, permettere la distribuzione di semplici files, tracciare gli accessi ai laboratori per valutare l'esposizione a rischi, gestire la suddivisione delle spese telefoniche, gestire le autorizzazioni all'accesso ai locali del DIFA al di fuori degli orari di apertura e perfino tracciare i tag per il controllo accessi. Come se non bastasse, sono stati anche integrati diversi applicativi esterni (MRBS per la gestione delle prenotazioni, OsTicket per tracciare le richieste degli utenti, MediaWiki per documentazione interna, e UniversityPlanner viene usato come datasource per alcune slide).

Per non farci mancare nulla, gli utenti delle altre strutture ospitate nel plesso devono poter accedere al sistema con le proprie credenziali (non UniBO) e senza che sia necessario da parte della loro struttura la configurazione di alcun servizio aggiuntivo.

Certo, visto con gli occhi di oggi, con a disposizione tanti framework ben collaudati e sistemi di autenticazione federata sembra un progetto semplice... Ma 6 anni fa la situazione era piuttosto diversa.

Indice

Elenco delle figure

Capitolo 1

Raccolta dei requisiti

1.1 Richiesta iniziale

La richiesta iniziale prevedeva la realizzazione di un sistema di ausilio alla redazione dell'elenco telefonico del nuovo Dipartimento di Fisica e Astronomia (DIFA).

Quasi immediatamente veniva evidenziata la necessità che per ogni persona fossero indicati, oltre il numero di telefono, anche la stanza, la buchetta (in cui la portineria deve inserire la corrispondenza in arrivo per la persona) e la mail.

1.2 Richieste aggiuntive

Parlando con la persona che curava il vecchio elenco, sono emerse ulteriori necessità:

- Tenere traccia della "carriera" del personale UniBO
- Censire il personale degli altri Enti presenti nelle strutture

- Individuare il personale esposto a rischi, per la redazione del documento sulla sicurezza

Visto che entrambi i vecchi Dipartimenti utilizzavano MRBS[1] per la gestione delle prenotazioni, ho proposto che venisse integrato nel portale per avere un unico "punto di accesso" per tutti ed evitare la frammentazione in una miriade di sitarelli specifici.

Già che veniva incluso MRBS, si è visto che avrebbe fatto comodo anche avere una gestione centralizzata dei display informativi presenti o di prossima installazione, con generazione automatica dei contenuti di alcune slide a partire dalle prenotazioni.

Il sistema del portale doveva quindi permettere una facile espandibilità, con la possibilità sia di linkare siti esistenti che di aggiungere sezioni (eventualmente ad accesso ristretto a determinati gruppi).

Ulteriori estensioni previste o in corso di realizzazione:

- generazione di un report multi-pagina per la suddivisione delle spese telefoniche dei vari gruppi di ricerca sui relativi fondi partendo da un foglio Excel coi dati del traffico, da incrociare coi dati presenti nel DB (procedura attualmente gestita a mano con enorme perdita di tempo!).

- gestione dei tag RFID per il controllo accessi ed eventuale interfacciamento al relativo database; eventuale integrazione con CIP[2] se tecnicamente e "politicamente" fattibile

- gestione dei codici delle chiavi[3] e delle schede di manutenzione

[1] Multi Room Booking System, http://mrbs.sourceforge.net

[2] Controllo Ingresso Persone, il sistema di controllo accessi con badge UniBO.

[3] Questa è una parte molto critica, che richiederà uso pesante di crittografia e/o altri accorgimenti per evitare che una eventuale compromissione del server fornisca all'attaccante i codici per potersi creare una chiave per ogni porta del Dipartimento!

- gestione delle risorse di rete, dell'inventario delle machine, dei ticket di assistenza tecnica e degli acquisti – per questo probabilmente converrà integrare OTRS[4] nell'infrastruttura del portale; potrebbe tornare utile anche per gestire altri flussi di lavoro (acquisti dell'Amministrazione, pratiche relative alla telefonia, ecc.)

Si è anche integrato un wiki[5] per raccogliere la documentazione tecnica interna e aiutare la memoria. L'accesso è ristretto al solo gruppo dei tecnici, salvo alcune pagine che avrebbero potuto/dovuto essere pubbliche ma che non sono mai entrate 'in produzione'.

[4]Software Open Source per gestione help desk, rimasto testa a testa con Remedy quando si valutava il sistema di ticketing per assistenza.cesia. [https://otrs.com/] Nell'implementazione attuale viene usato OsTicket [https://osticket.com] che, seppure più limitato, è stato di integrazione più rapida.

[5]Con MediaWiki [https://mediawiki.org]

Capitolo 2

Progettazione

2.1 Database

Non è stato possibile effettuare una progettazione formale e completa prima di iniziare la realizzazione del sistema, ma sospettando future richieste di espansione ho cercato di mantenere la massima modularità, come si può facilmente notare dal grafo di relazioni tra le tabelle.

Per la progettazione della struttura sono partito dall'entità "persona", che mi pareva (giustamente, posso dire a posteriori) centrale.

Da tale entità si diramano le relazioni con le altre entità. Le relazioni che non coinvolgono "persona" sono poche e spesso risultano semplici ausili per non sbagliare nell'inserimento di riferimenti (p.e. nome della struttura di appartenenza), tanto che a livello di interfaccia non si è ritenuto necessario implementarne la modifica.

La gestione delle slide forma una sorta di "isola" nel "mare" delle tabelle, dato che non si relaziona (al momento) con altre entità collegate a "persona"[1]. Potrebbe effettivamente risiedere su un database separato senza alcun

[1]Fare in modo che MRBS usi un riferimento a "persona" invece del nome di login quan-

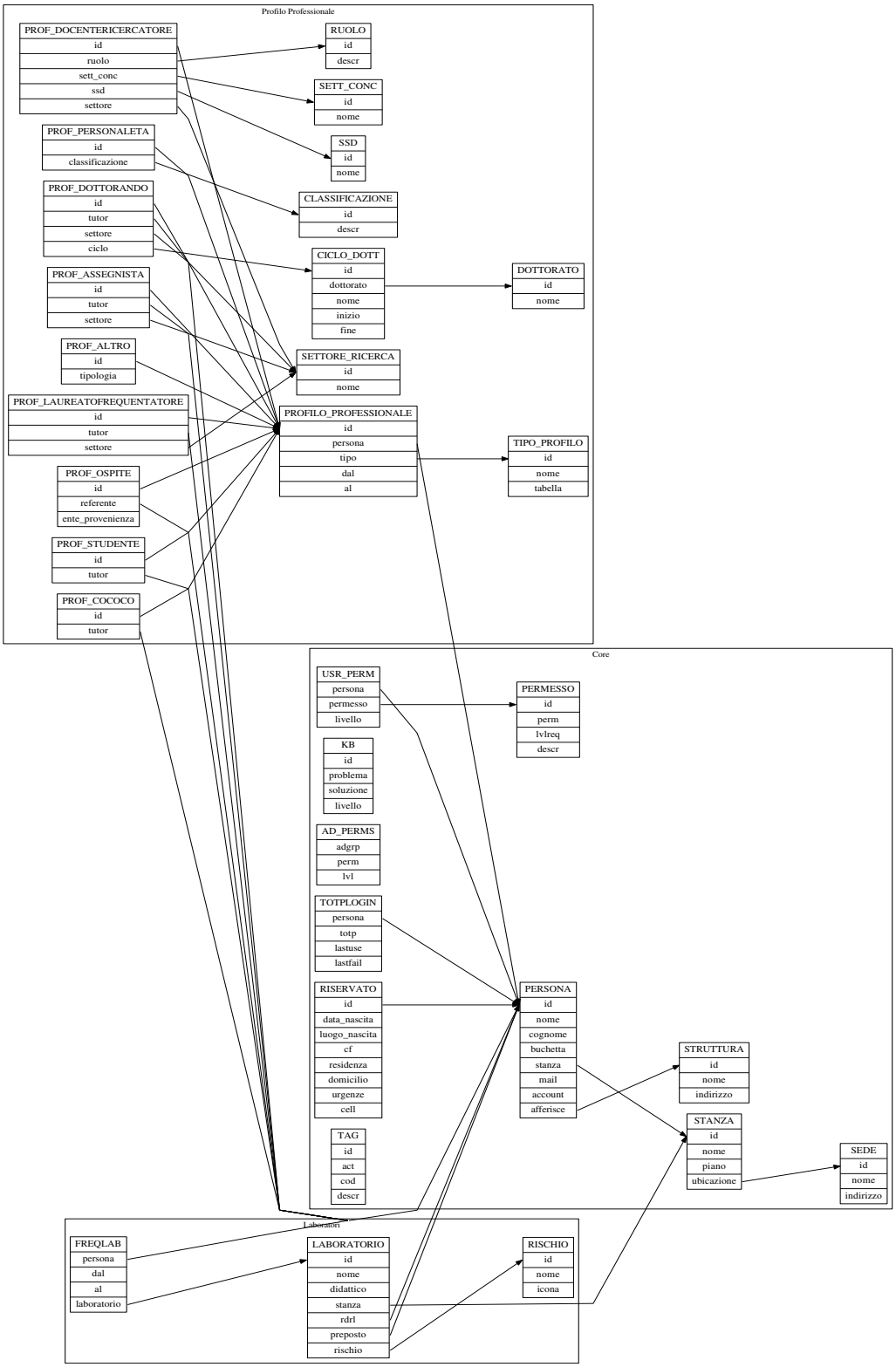

Figura 2.1: Diagramma relazioni globale (pt.1)

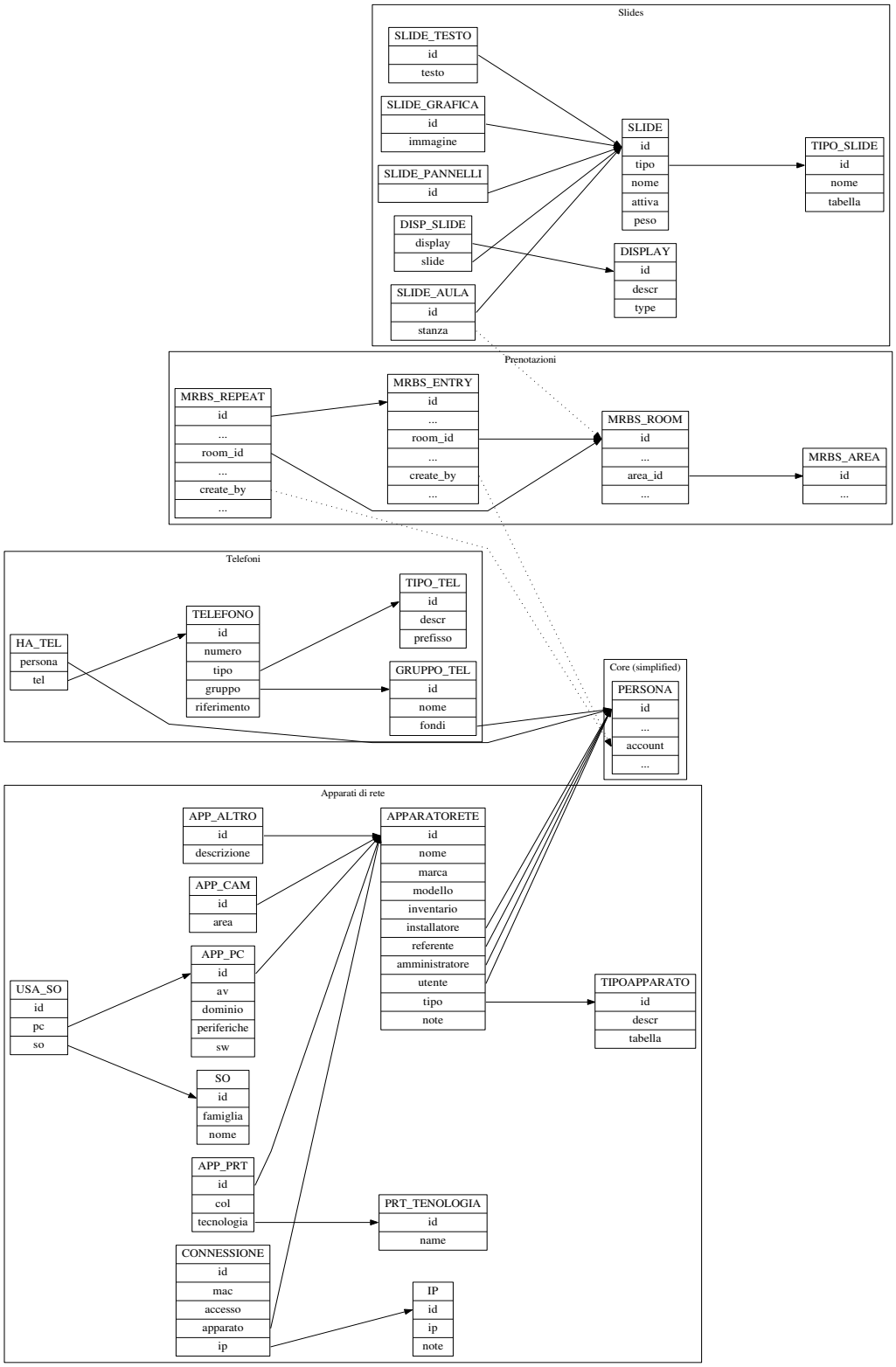

Figura 2.2: Diagramma relazioni globale (pt.2)

problema. Nel grafo delle relazioni, queste 'relazioni deboli' (non gestite con vincoli SQL ma solo a livello applicativo e denormalizzate) sono rese con linee tratteggiate.

Per ogni entità viene generato un id univoco per ottimizzare l'indicizzazione e permettere, generalmente, la modifica di ogni altro campo:

```
CREATE TABLE `tablename` (
    id NOT NULL PRIMARY KEY AUTO_INCREMENT,
    ...
    )
```

2.1.1 Il nucleo (core)

Questa è la sezione "più vecchia" dell'archivio, contenente le prime tabelle implementate.

Contiene ovviamente la tabella PERSONA e le altre necessarie per l'autorizzazione, le informazioni 'logistiche' (STANZA e SEDE) e quelle di afferenza (STRUTTURA).

Ho lasciato in core anche la tabella TAG, anche se non ha attualmente relazioni. Potrebbe tornare utile se ci sarà collaborazione da parte del CeSIA per interfacciarsi a CIP ed abilitare/disabilitare così i tag degli utenti senza dover usare una procedura separata.

Una particolarità che ritengo di dover far notare: nell'implementare l'entità "persona" ho utilizzato due tabelle invece di una, legate da una relazione 1:1 opzionale e che utilizzano lo stesso ID. La seconda tabella, infatti, mi serviva per raccogliere campi "riservati" dell'entità "persona": creandola come tabella a parte, nel caso un domani si cambi il backend, ci potrà permettere

do si crea una prenotazione richiederebbe una serie di modifiche che mi paiono decisamente sproporzionate, anche in considerazione della manutenibilità futura del sistema.

di limitare in modo più forte l'accesso a tali dati (per es. realizzando un modulo del portale che acceda con un diverso utente al database). In pratica si tratta dello stesso sistema utilizzato per implementare la derivazione/specializzazione di oggetti (istanze di entità) ma senza un campo in "persona" che indichi la presenza o meno dei dati riservati: per determinare se ci sono, è necessario fare una query aggiuntiva. Ciò non crea problemi di performance, anzi le migliora, dal momento che l'accesso ai dati riservati è saltuario (pochi acessi al giorno) mentre l'accesso ai dati generici della persona è frequentissimo (parecchie centinaia di accessi l'ora) ed in questo modo è più facile che le righe rimangano nella pagecache di MySQL.

2.1.2 Un modulo complesso: le procedure

La gestione delle procedure è stato, finora, il modulo più complesso e di difficile analisi, soprattutto perché non è stato sempre chiaro che cosa si volesse modellare.

La richiesta iniziale era "*tenere d'occhio lo stato degli acquisti*". Prima possibile soluzione: una semplice scheda per l'entità PRATICA (meglio tenersi sul generico) con alcuni campi (attributi) da compilare. Soluzione scartata perché non permetteva di tracciare il flusso del lavoro.

Inizia quindi a chiarirsi leggermente la situazione: una PRATICA di acquisto è istanza di una PROCEDURA composta da diversi passi (STEP)[2].

- arriva una richiesta
- si fa un preventivo
- si richiedono CIG e CUP

[2]Non aggiornata dopo le ultime modifiche normative che richiedono ben due decreti del Direttore per ogni acquisto...

- si fa l'ordine in Consip
- arriva la merce
- arriva la fattura (spesso molto dopo l'arrivo del materiale)
- si inventaria
- si collauda
- se si tratta di materiale che deve poter essere portato fuori dal Dipartimento si compila e si fa firmare il modulo di affidamento
- si consegna il materiale
- quando è il momento si paga la fattura
- finalmente si può chiudere la pratica

E già si può notare che un "semplice" acquisto di materiale informatico richiede il lavoro di almeno due gruppi di persone: i tecnici e gli amministrativi. E quindi si introduce la nozione di GRUPPO di lavoro (un GRUPPO può gestire molti STEP, ed ogni STEP è gestito da esattamente un GRUPPO).

E fin qui non sarebbe un grosso problema. È ancora tutto bello lineare. Ma quando c'è di mezzo la burocrazia (o, più semplicemente, il tentativo di modellizzare situazioni reali), le cose si complicano rapidamente.

Intanto non è detto che si riesca a soddisfare ogni richiesta con un solo ordine. Poi, come ci è capitato, un ordine potrebbe venire rifiutato dal venditore richiedendo quindi un "riavvio" della pratica. In certi casi deve essere possibile inventariare dalla bolla, e talvolta nella stessa richiesta c'è materiale sia inventariabile che non.

Un esempio concreto di richiesta non banale (ma ispirata ad un caso realmente accaduto) che mette alla prova la sanità mentale del sistema:

> "*Dovrei acquistare un portatile. Voglio che uno dei due dischi interni venga sostituito con un SSD. Inoltre mi serve una cartuccia di toner per la mia stampante.*"

Il toner non sarà da inventariare (si tratta di materiale di consumo). Viene fatto l'ordine del portatile ad un fornitore, ma non arriva per una serie di scuse più o meno credibili. Quindi ordine da annullare in MEPA e rifare. Il disco SSD è fornito solo da un altro venditore, quindi richiede un ordine separato. Visti i ritardi, il materiale sarà da consegnare il prima possibile, e non sarà quindi possibile aspettare la fattura per inventariare.

Ed ecco che una bella procedura lineare si è intricata non poco.

Intanto è emerso il caso del materiale di consumo, che viene gestito a parte ma deve comunque essere considerato nella procedura. Inoltre i due flussi di lavoro sono spesso molto simili, ma i passi possono non essere nella stessa sequenza. Inoltre certi passi richiedono che siano disponibili informazioni da passi precedenti (per impedire che vengano prese scorciatoie: l'annullamento dell'ordine fa ritornare al preventivo ma non sarà necessario richiedere nuovamente CIG e CUP, ma non deve essere possibile ne saltare la richiesta di CIG e CUP se non c'è stato un annullamento dell'ordine, ne effettuarne una seconda se c'è stato annullamento! Dovrà però essere possibile modificare il solo CUP se la cifra necessaria è aumentata, dato che potrebbero non essere più sufficienti i fondi a cui faceva riferimento il vecchio CUP).

Quindi per ogni passo deve essere possibile definire vincoli: se (non) esistono nell'istanza della procedura uno o più passi (eventualmente ognuno con determinati attributi) va impedita la creazione del passo in esame.

Quindi si può dire che:

- una RICHIESTA viene gestita tramite una o più pratiche

- una PRATICA è l'istanza di una PROCEDURA
- una PROCEDURA può essere di diversi tipi (es: ACQUISTO MATERIALE INVENTARIABILE, ACQUISTO BENI DI CONSUMO, ASSITENZA TECNICA[3])
- una PROCEDURA è composta da uno o più passi (STEP)
- uno STEP è associato ad una o più PROCEDURE
- uno STEP è gestito da un gruppo di lavoro (WORKGROUP), composto di PERSONE
- uno STEP può avere diversi CAMPI
- un CAMPO può essere in uno o più STEP[4]
- (spoiler) introduco una pseudo-entità PROCSTEP per evitare relazioni ternarie da dover semplificare successivamente
- per ogni PRATICA è necessario tenere uno storico denormalizzato (PRATICASTEP) di tutte le istanze di ogni STEP effettuato per la PROCEDURA seguita (qui torna comodo PROCSTEP), col valore (ha_valore) di ogni campo assegnatogli
- la creazione di un PRATICASTEP può essere limitata da un numero arbitrario di regole (CONSTRAINT), che devono essere tutte soddisfatte[5]

[3]Giusto per considerare qualcosa di completamente diverso da un acquisto

[4]Non entro ulteriormente nel dettaglio della struttura del CAMPO perché non riguarda direttamente il database ma piuttosto l'interfaccia.

[5]Mi pareva eccessivo implementare un albero AND-OR, dal momento che per simulare l'OR è comunque possibile creare altri PROCSTEP relativi alla stessa PROCEDURA e allo stesso STEP ma con diverso set di CONSTRAINT.

- un CONSTRAINT può essere di (non) esistenza di un determinato PROCSTEP[6] o di relazione di un valore di CAMPO di un precedente PROCSTEP[7] con una costante (p.e. acquisti di valore inferiore a 100€ non vanno inventariati)[8]

Si arriva quindi al diagramma E-R in figura 2.3.

Dovrebbe risultare sufficientemente versatile da rappresentare ogni possibile procedura, evitando la duplicazione degli STEP (con relativi CAMPI) tra diverse procedure. Inevitabilmente, per ogni procedura andranno reinseriti i CONSTRAINT su ogni STEP: logicamente, procedure diverse possono avere (e generalmente avranno) requisiti diversi (anche se simili) per permettere l'accesso ad uno stesso STEP.

Quello che apparentemente manca è un metodo per ordinare gli step. Effettivamente in una prima analisi si era previsto una auto-relazione 'next' 1-a-N di STEP, ma con l'inserimento dei CONSTRAINT questa necessità è venuta a mancare: l'ordinamento degli step viene dato dai constraint stessi, che 'istruiscono' l'interfaccia affinché presenti solo gli step permessi, scartando gli altri. In pratica si permette la creazione di un PRATICASTEP solo se relativo ad un PROCSTEP senza CONSTRAINT o con tutti i CONSTRAINT verificati.

Intuitivamente, la gestione dei CONSTRAINT risulterà parecchio pesante sia per il DB che per il codice applicativo. Infatti, per ogni "candidato" PROCSTEP da proporre all'utente, è necessario eseguire vari controlli.

[6]Implicito: nella pratica corrente, quindi di esistenza di un PRATICASTEP relativo al PROCSTEP dato

[7]Idem

[8]Non si vuole considerare un vincolo del tipo "è richiesta la verifica del CUP se il nuovo preventivo è superiore al vecchio" per l'eccessiva complicazione che introdurrebbe, a fronte di una limitata utilità.

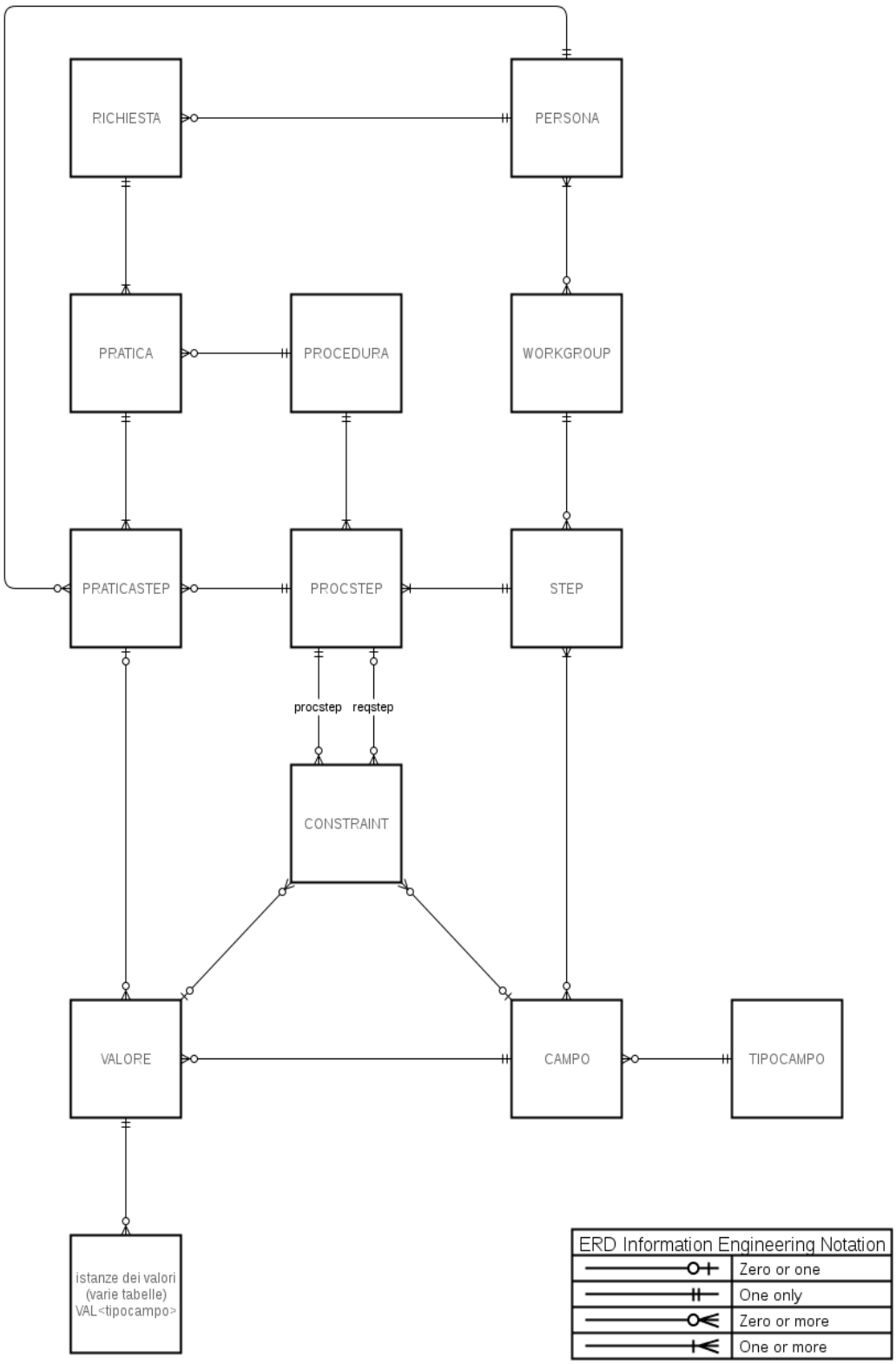

Figura 2.3: Diagramma ER della procedura pratiche

Implementativamente, si può notare che solo due relazioni sono N-a-M (WORKGROUP-PERSONA e STEP-CAMPO): per queste vanno introdotte due tabelle di relazione (rispettivamente MEMBRO e PARAMS) aventi come campi le foreign key delle tabelle che mettono in relazione. Tutte le altre relazioni saranno implementate usando un campo foreign key nell'entità "dipendente".

Al termine del lavoro di progettazione e avanzato stadio di implementazione, però, è stata presa la decisione "politica" che il sistema fosse troppo rigido poiché obbligava a formalizzare (e seguire SEMPRE, senza eccezioni a meno di non formalizzare anche queste!) le procedure, quindi si è optato per un sistema generico di gestione ticket (OsTicket) con personalizzazioni minime, tutt'ora in uso. Per questo nel sistema attualmente online purtroppo non c'è traccia della gestione delle pratiche.

2.2 PHP: generazione pagine

Il codice è suddiviso in vari livelli. Il livello più esterno è praticamente solo di collante e si occupa di:

- inizializzare l'ambiente
- inizializzare il dynamic class loader
- inizializzare le singole applicazioni
- richiamare l'applicazione corrente
- gestire il popolamento del template per la pagina finale

Ogni applicazione viene gestita in due fasi:

- init : viene determinato se l'applicazione può essere attivata o meno – il costruttore aggiunge self ad un array se l'applicazione può essere eseguita; si è scelto di non utilizzare una object factory per mantenere la massima flessibilità[9]
- run : se l'applicazione è quella attualmente selezionata, la esegue

2.3 PHP: interfacciamento al database

Per ogni tabella che rappresenti un'entità è stata realizzata una classe, derivata da dbdata, che implementa i metodi utili per tutte le altre classi e stabilisce l'interfaccia minimale che deve essere implementata.

Il concetto è che ogni entità deve sapere come presentarsi su una pagina per la visualizzazione piuttosto che per la modifica, e deve anche sapere come aggiornarsi all'interno del database ricevendo il form che lei stessa ha contribuito a generare.

Questo ha portato, talvolta, a dover ricorrere a strani compromessi. Un esempio è la gestione delle slide per i televisori:

- la classe slide, quando deve "presentarsi" per la sola lettura, si limita a richiamare la visualizzazione dell'oggetto derivato
- il visualizzatore e l'editor di slide devono essere a conoscenza di come visualizzare le proprietà di slide, per poterle impaginare correttamente in una tabella

Purtroppo, tutte le soluzioni più "pulite" provate portavano rapidamente a codice molto più pesante e meno manutenibile.

[9]Varie applicazioni usano un sistema analogo per caricare moduli, e questo avrebbe richiesto una object factory per ogni applicazione, quando il dynamic class loader di PHP fa egregiamente il suo lavoro.

Inoltre, la maggior parte delle classi derivate da dbdata[10] implementa la creazione della propria tabella nel caso non esista ancora:

```
class config extends dbdata {

    public function __construct($db) {
        try {
            parent::__construct($db);
        } catch(MissingTableException $e) {
            // Define table
            $q="create table `".get_class($this)."` (".
                "`id` char(10) primary key, ".
                "`val` text NULL, ".
                ") ENGINE = MyISAM ".
                "COMMENT = 'Configurazione e stato';";
            $db->query($q);
            parent::__construct($db);    // Retry
        }
    }
...
}
```

In questo modo il passaggio da ambiente di test ad ambiente di produzione non richiede la preventiva creazione delle nuove tabelle. Nei rari casi in cui invece venga modificato lo schema di tabelle esistenti è necessario un intervento manuale (probabilmente con qualche minuto di downtime).

2.3.1 Join tra tabelle

L'aspetto peculiare dell'astrazione del database che ho creato è che prevede esplicitamente il join tra tabelle (gestito dalla classe dbdata).

[10] Quelle del DB principale, non dei pacchetti di terze parti.

L'implementazione non è stata banale, in quanto tabelle diverse potrebbero contenere campi omonimi, e più campi di una tabella potrebbero essere foreign key per la stessa tabella.

Si è risolto il problema creando ricorsivamente nomi di campo univoci e rappresentativi del 'percorso', mantenendo così una buona mnemonicità oltre ad una 'quasi-garanzia' di univocità.

Un esempio per chiarire:

Tabella A:

- id primary key
- testo varchar

Tabella B:

- aref foreign key references A.id
- altro int

Potrò quindi usare codice tipo:

```
$b=new B($db);
$b->leftjoin('A', 'aref', 'id');
$tmp=$b->get_arr();
print_r(array_keys($tmp[0]));
```

ed otterrò:

```
array (
    [0] => b_aref
    [1] => b_altro
    [2] => b_aref_id
    [3] => b_aref_testo
)
```

In pratica il campo aref viene 'espanso' con l'aggiunta di underscore e dei nomi dei campi della tabella linkata. In caso due campi puntino la stessa tabella, i nomi generati rimarranno comunque univoci dato che non posso avere campi omonimi.

Un problema potrebbe essere rappresentato da una situazione "patologica" in cui in una tabella il campo 'a_ref' fa riferimento alla tabella b che contiene il campo 'bad', e il campo 'a' che fa riferimento alla tabella c che contiene il campo 'ref_bad', ottenendo una omonimia per 'tbl_a_ref_bad'.

Tale errore non è facilmente eliminabile poiché i nomi vengono generati ricorsivamente, ma non crea problemi se se ne tiene conto quando si danno i nomi ai campi. Alternativamente, andrebbe scelto un carattere diverso da '_', compatibile con la sintassi SQL e sufficientemente poco utilizzato, ma questo sposterebbe solo il problema, senza risolverlo.

2.3.2 Filtraggio ed ordinamento

La classe dbdata prevede due metodi (filter ed order) molto simili. Entrambi si aspettano come parametro un array che indichi i campi su cui devono agire ed entrambi ritornano l'array precedentemente impostato (per poterlo ripristinare).

Per filter si possono usare due notazioni: specificando per un elemento sia la chiave che il valore, nella clausola where la chiave sarà usata come nome di campo (eventualmente opportunamente 'decorato') da confrontare per l'uguaglianza con il valore dato. Se (come spesso succede) serve una query più complessa o non di semplice uguaglianza, allora basta specificare il solo valore, senza chiave, e questo sarà aggiunto as-is (non 'decorato'!) nella clausola where. Tutti gli elementi del filtraggio sono uniti da AND. Per clausole where con OR è necessario usare la seconda notazione. Bisogna

prestare molta attenzione nell'uso della seconda notazione poiché, venendo inserita direttamente, deve utilizzare i nomi delle tabelle 'decorati' ed i nomi dei campi non 'decorati' (limitazione della sintassi SQL).

Una ulteriore difficoltà rilevata nell'uso è l'ordinamento(/filtraggio) secondo un campo di una tabella joinata: è necessario costruire attentamente l'elemento dell'array da passare ad order()/filter(), poichè purtroppo non risulta assolutamente intuitivo.

Un esempio, anche in questo caso, può risultare chiarificatore.

La tabella mrbs_room contiene un campo area_id che è foreign key per mrbs_area. Per joinare le due tabelle ed avere poi i risultati ordinati prima per mrbs_area.area_name e poi per mrbs_room.room_name, uso[11]:

```
$this->aule=new mrbs_room($db);
$this->aule->leftjoin('mrbs_area', 'area_id', 'id');
$this->aule->order(array("`area_id`.`area_name`",
    'room_name'));
```

Da notare la chiusura del backtick prima del punto nel primo elemento (il nome della tabella, 'decorato') e la sua riapertura dopo di esso (il nome del campo nella tabella, non 'decorato').

2.4 Codice lato client (HTML e Javascript)

Non è stato utilizzato AJAX poiché avrebbe più che raddoppiato lo sforzo per la scrittura dell'applicazione (che deve risultare utilizzabile anche se JavaScript è disattivato[12]), senza particolari vantaggi nell'usabilità.

[11] Dal costruttore di slide_aula.php

[12] Questa è un'altra delle cose MOLTO cambiate in 6 anni... Oramai il codice JS è molto più cross-browser... Chi ha sviluppato web app cross-browser e client-side ai tempi di IE6 sa bene di cosa parlo!

Viene usato un minimo di JavaScript, principalmente di terze parti, per manipolare l'estetica di parti della pagina (come aprire e chiudere una lista ad albero: senza JS risulta completamente aperta, ma questo è comunque il default e non crea problemi di usabilità).

Rimane comunque possibile aggiungere maggiore interattività in una seconda fase.

Sperimentalmente, si è aggiunto il modulo DataTable per jquery nella schermata della lista del personale: l'estetica e l'usabilità per gli utenti "comuni" sono migliorate (possibilità di riordino della tabella, casella di filtraggio sulla pagina), ma i tempi di caricamento si sono drasticamente allungati e gli utenti "avanzati" sono penalizzati (il ctrl-f non funziona più, tanto per fare un esempio). Però la cosa è piaciuta (evidentemente gli utenti "avanzati" sono un'esigua minoranza) e si è deciso lasciarlo.

Capitolo 3

Implementazione

Per l'implementazione mi sono basato sul collaudatissimo stack LAMP[1], creando una coppia di vhost sui server web di Dipartimento (un vhost per lo sviluppo ed il testing, un altro per la "produzione"). Il carico previsto non fa ritenere necessario l'uso di un server dedicato[2].

Sebbene l'uso di un backend di database con funzioni più avanzate potesse apparire auspicabile (p.e. le funzioni di inheritance in PostgreSQL sarebbero tornate molto comode per la "gerarchia" dei profili professionali), questo avrebbe portato ad una maggiore complicazione per la manutenzione (backup, gestione della sicurezza, clustering) e ad una minore (o nulla) portabilità verso altri backend. Usando solo i normali vincoli di integrità referenziale e la generazione automatica di numeri di sequenza non si dovrebbero avere problemi nel caso un domani risultasse necessario un cambio di backend.

[1] Linux, Apache, MySQL, PHP

[2] Effettivamente ha sempre funzionato in una VM abbastanza ridotta e non ha praticamente mai saturato le risorse allocate.

3.1 Interfacciamento al database

L'astrazione dell'interfacciamento al db è su due livelli: a quello più basso ho posto un "thin wrapper" per la classe mysqli, mentre per quello intermedio ho creato la classe dbdata, da usare come base per le classi che rappresentano le tabelle. In molti casi, la classe della tabella sarà vuota e tutto il lavoro sarà effettuato dalla classe dbdata.

La classe db (il thin wrapper) si occupa principalmente di gestire gli errori e convertire i result set in array. Sarà probabilmente espansa per gestire il caching tramite memcached.

La classe dbdata è in grado di gestire la generazione automatica delle query per filtrare, ordinare ed effettuare join (per il momento servivano solo left join, ma non c'è problema ad implementarne altri tipi).

Ho successivamente aggiunto altre due classi "sopra" a dbdata per gestire le relazioni molti-a-molti in modo più semplice e senza dover ogni volta reimplementare l'aggiunta e la cancellazione di righe alla tabella di relazione. Tali classi sono 'relation' ed 'attributed_relation' (con la seconda che estende la prima). 'relation' implementa un semplice metodo di redirect per la gestione delle richieste di aggiornamento, mentre 'attributed_relation' offre un'interfaccia per gestire in modo semplificato ed omogeneo la generazione dei campi nei form HTML. Questa modifica richiederà una riscrittura parziale del codice che gestisce le persone (per es. la parte relativa a stanze e telefoni), ma l'usabilità dovrebbe guadagnarne parecchio.

Sono ancora in dubbio se implementare o meno un'astrazione per la multi-table-inheritance usata per le gerarchie, ma probabilmente sarebbe una complicazione inutile.

3.2 Autenticazione ed autorizzazione

3.2.1 Autenticazione

Il sistema richiede le credenziali con un normale form HTML. Essendo la password istituzionale un dato prezioso, ovviamente la pagina di login è accessibile esclusivamente tramite HTTPS con un certificato ottenuto tramite il GARR.

A seconda del dominio indicato nello username, viene selezionato il backend di autenticazione da utilizzare:

- unibo.it, studio.unibo.it, esterni.unibo.it: bind al server LDAPs
- altri domini autorizzati: tentativo di autenticazione al server di posta dell'Ente tramite IMAPS o POP3S a seconda dei casi

2FA

Per migliorare la sicurezza, ho previsto la possibilità di usare autenticazione a due fattori TOTP (p.e. tramite l'app Google Authenticator). Agli utenti che la attivano viene impostata la capability '2fa' al momento del login.

Per l'implementazione della 2FA ho fatto uso sia della libreria Google2FA[3] che della libreria PHPQRCode[4]. È stato necessario qualche piccolo "ritocco" in entrambe: per Google2FA ho reimplementato il controllo del codice TOTP per prevenire replay attack, mentre in PHPQRCode ho modificato l'output dell'immagine col codice in modo da poterla inserire inline all'interno della pagina, come stringa base64-encoded.

[3] http://www.idontplaydarts.com/2011/07/ga.php_.txt
[4] http://phpqrcode.sourceforge.net/

Per sicurezza, il segreto alla base della generazione del codie TOTP non verrà più visualizzato[5] e sarà solo possibile eliminarlo. In questo modo chi abbia accesso ad una sessione lasciata incautamente aperta non potrà effettuare un nuovo login neanche disponendo delle credenziali della "vittima".

Sviluppi futuri: SAML?

Non si è ancora implementato SSO[6] per questioni di tempistica: la parte "base" dell'applicativo (gestione delle anagrafiche), come sempre, serviva per 'ieri'. Una volta che il sistema sia a regime (ovvero che non vengano continuamente aggiunte richieste di nuove funzionalità) si potrà avviare l'iter richiesto dal Ce.S.I.A. per sostituire l'autenticazione/autorizzazione proprietaria con SSO[7].

Per il personale degli altri Enti si è scelto di utilizzare il relativo server di posta per fornirgli comunque un metodo di autenticazione. Se poi si deciderà di estendere il sistema con SAML, sarà da implementare l'integrazione coi loro sistemi.

Il sistema attuale infatti, anche se ben funzionante, ha un enorme difetto: costringe l'utente a fornire le proprie credenziali ad un sito potenzialmente inaffidabile (un sito per lui 'non istituzionale', in cui un amministratore o un hacker potrebbero aver installato un sistema di logging delle credenziali).

Per evitare questo, ed avere comunque accesso all'identità verificata dell'utente, l'Ateneo utilizza SAML[8] che (prima o poi) dovrebbe permettere SSO anche per l'accesso alla webmail. Visto che SAML supporta nativamen-

[5] Purtroppo nel DB rimane in chiaro...

[6] Single Sign On

[7] Questo è poi stato fatto, ma solo in ambito di test. Presenta diversi problemi e pochi vantaggi.

[8] Security Assertion Markup Language

te più IdP[9], appare una scelta naturale. Eventuali Enti che non abbiano un IdP possono comunque essere gestiti utilizzando un IdP ad-hoc che utilizzi, come oracolo di autenticazione, il server mail. Si tratta però di un sistema piuttosto complesso, rigido e che non si "sposa" bene con ISPConfig (usato per gestire i virtualhost).

3.2.2 Autorizzazione

La distinzione del backend di autenticazione si è resa necessaria per poter ottenere in modo automatico i gruppi di appartenenza dell'utente in DSA[10] e semplificare l'inserimento dati e la manutenzione delle autorizzazioni (l'appartenenza ad alcuni gruppi viene gestita direttamente dall'Ufficio Personale).

Per fortuna il tree LDAP di AD permette le ricerche ricorsive usando una sintassi particolare: (member:1.2.840.113556.1.4.1941:={$udn}) Purtroppo si tratta di una query molto lenta, che se applicata all'intero albero richiede facilmente più di due minuti. Questo ritardo, in fase di login, risulta assolutamente inaccettabile. Ho quindi dovuto selezionare alcuni rami ed eseguire più volte la ricerca. Riducendo lo scope, in meno di un secondo si riesce ad avere la risposta. Altro problema: {$udn} non è noto a priori. Ho dovuto quindi effettuare prima una ricerca per ottenerlo partendo dall'indirizzo mail.

Al personale UniBO potranno quindi essere assegnati automaticamente alcuni gruppi (definiti nella tabella 'ad_perms').

Per tutti, inoltre, sono definiti dei gruppi a gestione manuale (tramite le tabelle 'permesso' ed 'usr_perm'), in aggiunta a quelli assegnati automaticamente.

[9]Identity Provider, ovvero server che possono verificare le credenziali dell'utente.

[10]Directory Service d'Ateneo

Per gestire l'autorizzazione ho realizzato un'apposita classe ECAPS (Extended CAPabilities Set) in PHP, affiancata da una classe helper ACL (Access Control List) che "riunisce" quattro ECAPS per le operazioni CRUD[11].

Una 'capability' può essere considerata alla stregua di un gruppo, con in più la possibilità di essere negata.

La 'extended capability' aggiunge la possibilità di specificare un livello.

Esempi:

cap capability semplice: soddisfatta se l'utente la possiede

cap=3 capability con livello: soddisfatta se l'utente la possiede con un livello pari o superiore a quello indicato (3, in questo caso)

-cap capability negativa: soddisfatta solo se l'utente non la possiede

La capability negativa con livello, pur essendo possibile, non avrebbe interpretazione univoca, e quindi il livello viene semplicemente scartato.

La rappresentazione normale di ECAPS è una stringa formata da capabilities separate da virgola.

La classe ECAPS emette un warning se si tenta di impostare un set di capability che non può essere soddisfatto (p.e. *cap,-cap*) ed usa un'impostazione di tipo fail-safe (ovvero non permette l'accesso).

Ogni operazione può essere protetta con ECAPS o ACL.

Le classi ECAPS ed ACL sono anche state utilizzate per migliorare la granularità del controllo accessi in MRBS (branch mrbs_acl).

[11] Create, Read, Update, Delete

3.3 Esecuzione periodica e configurabilità a runtime

Per alcuni moduli (p.e. quello relativo alle richieste) si è visto che sarebbe stato utile avere un'esecuzione periodica, indipendentemente dalle altre attività del sistema. Contestualmente si è visto che molto spesso sarebbe stato utile poter modificare alcune opzioni senza dover intervenire sul codice sorgente.

Per questo sono state create le interfacce 'cronnable' e 'configurable'.

3.3.1 Interfaccia cronnable

```
interface cronnable {
    // Called for cron events
    // Must not output anything (output would end up in
    // cron's syslog!)
    // Returns NULL if all went well, an error message
    // for failures
    public function cron_run($last, $now);
}
```

La classe che implementa cronnable deve prevedere un controllo di accesso esplicito per le invocazioni tramite cron.

Le capability assegnate all'utente virtuale usato per l'esecuzione di cron sono configurabili (vedi interfaccia configurable) e come default viene usato 'login,cron'. La capability 'cron' non può essere assegnata in altro modo e questo previene la possibilità che un utente ne abusi.

cron_run() riceve i timestamp dell'ultima esecuzione e quello attuale, così da poter decidere se deve o meno eseguire un'azione.

$now è sempre almeno un minuto più avanti di $last, ma non è garantito che lo sia di un solo minuto: in caso di problemi (o se viene ripulito il db della configurazione) può esserci un salto di giorni. Avendo entrambi i timestamp l'app può decidere se fare o meno l'azione schedulata. Un esempio può essere la generazione di un report giornaliero delle presenze: in caso di blocco del sistema, alla riattivazione potranno essere generati i report per tutti i giorni intermedi.

3.3.2 Interfaccia configurable

```
class ConfigData {
    // An integer
    const TYPE_INT='i';
    // A simple string
    const TYPE_STRING='s';
    // An integer representing date & time in Unix format;
    // gets rendered as human-readable string
    const TYPE_TIMESTAMP='t';
    // List from a DB table
    // Must specify $params for register():
    // ['table'=>'tablename', 'disp'=>'colname',
    //  'key'=>'id', 'multiple'=>true, 'separator'=>' ']
    const TYPE_DBLIST='dl';
    // List from an array
    // Must specify $params for register():
    // [ 'data'=>[[$id,$disp], [$id,$disp]*],
    //  'multiple'=>false, 'separator'=>' ']
    const TYPE_LIST='l';

    const CFG_MC_HOST='Conf:mc_h';
    const CFG_MC_PORT='Conf:mc_p';
```

```
    private function __construct() { /* Disabled */ }
    private function __clone() { /* Disabled */ }
    private function __wakeup() { /* Disabled */ }

    public static function init($db) {
...
    }

    // Return the value associated with key $k, or
    // - $def if not found and $def is given
    // - registered default value if no default is given
    // - false if everything else fails
    // Value is typed if key is registered, string
       otherwise
    public static function get($k, $def=false) {
...
    }

    // Set value $v for key $k
    // Do not store in backend if $expire is set
    public static function set($k, $v, $expire=0) {
...
    }

    // Registrazione delle variabili per la configurazione
    // longdescr is a callback
    public static function register($k, $t, $descr,
        $default=NULL, $longdescr=NULL, $params=NULL) {
...
    }

    public static function listvars() {
...
```

```
    }

    public static function get_long_descr($k) {
...
    }

    // Pre-register self-defined entries
    private static
     $configtable=[
        self::CFG_MC_HOST => [
        'type'=>self::TYPE_STRING,
        'descr'=>'ConfigData: server memcached
           (indirizzo)',
        'default'=>NULL,
        'longdescr'=>[__CLASS__, 'get_long_descr'],
        'class'=>__CLASS__,
        'params'=>NULL,
        ],
        self::CFG_MC_PORT => [
        'type'=>self::TYPE_INT,
        'descr'=>'ConfigData: server memcached (porta)',
        'default'=>'11211',
        'longdescr'=>[__CLASS__, 'get_long_descr'],
        'class'=>__CLASS__,
        'params'=>NULL,
        ],
    ];
}

interface configurable {
    public static function get_conf_longdescr($key);
}
```

Il costruttore della classe (app o modulo) si occupa di richiamare ConfigData::register() per dichiarare le variabili che intende usare e che verranno aggiunte al database (se non sono già presenti).

Per semplificare alcune classi, è definita anche una helper class configurable_app che implementa un metodo register_conf richiamabile dal costruttore della classe derivata per 'registrare' le variabili di tutti i moduli della app stessa, fattorizzando il codice comune.

get_conf_longdescr() viene utilizzata dal modulo di gestione della configurazione per creare il popup di help nel caso l'utente lo richieda. Tale testo lo si poteva anche inserire in register(), ma in tal caso avrebbe usato molta memoria anche quando non necessario. Da notare che a ConfigData::register() per longdescr può essere passata una stringa o una funzione di callback, che può anche essere un metodo (statico, ovviamente) della classe che implementa l'interfaccia.

3.4 MRBS

Ad MRBS sono state apportate una serie di modifiche (già rilasciate in un branch apposito) per la gestione delle ACL. Sono previste ulteriori modifiche per rendere il sistema ancora più versatile, ma già ora è abbastanza semplice controllare chi può prenotare una stanza.

L'integrazione col resto del portale ha richiesto la realizzazione di un session handler di poche righe.

3.5 Slides per display informativi

Il Dipartimento ha in dotazione cinque display informativi a disposizione del pubblico (uno in Berti-Pichat 6/2, due nella sede 'storica' di via Irnerio,

uno in fase di installazione al Navile, uno in Ranzani 1 e che forse verrà dismesso).

Ogni display deve mostrare ciclicamente una serie di slide con diverse informazioni, come ad esempio l'orario delle lezioni, avvisi ed annunci, ecc.

L'ordine delle slide deve essere lo stesso su tutti i display, ma per ogni display deve essere possibile stabilire quali slide mostrare e quali no.

Per l'ordinamento si è scelto di assegnare ad ogni slide un peso (da 0 a 20) così che, abbinandolo al nome, non ci siano ambiguità[12]. Per stabilire quali slide mostrare su ogni display si implementa una relazione molti-a-molti tramite la tabella disp_slide.

A complicare (leggermente) le cose, i display possono avere risoluzione diversa (hd, hdready o sd, o addirittura potrebbero non avere capacità grafiche!). Di questo se ne occupa il dispatcher in secslides, selezionando i parametri più indicati per ogni risoluzione prevista. Il codice del metodo get_display() di ogni slide si occuperà di adattare, per quanto possibile, la rappresentazione a tali parametri. Questo metodo si è rivelato sufficientemente flessibile da essere usato sia dalla visualizzazione in full-HD (1920x1080) sia da quella usata nei thumbnail (128x128) con risultati (quasi) sempre buoni!

È possibile sia far "puntare" il display (se dispone di un browser) ad una pagina che visualizzerà ciclicamente ogni slide, sia scaricare tutte le slide in un'unica pagina, pronte per la post-elaborazione locale[13].

Ogni slide è strutturata in modo da ricordare la pagina web del Diparti-

[12] È bene ricordare che SQL non garantisce l'ordinamento dei risultati delle query, se non richiesto esplicitamente.

[13] Soluzione necessaria per l'uso sul WDTV-Live in Ranzani 1, che non dispone di un browser, anche se questo implica che l'orario visualizzato nella barra sotto l'header sarà sempre 'indietro' di un ciclo di aggiornamento, ovvero fino a 10 minuti (per ridurre il wear-out del flash drive utilizzato).

mento. Ci sarà quindi un header (ottenuto ridimensionando opportunamente la grafica presa dal sito del Dipartimento), una 'area menù' sulla sinistra in cui compariranno tutti i nomi delle slide (con la slide corrente evidenziata) ed un'area 'attiva' dove compariranno i contenuti specifici della slide. Header e menù sono gestiti dalla classe base 'slide', che implementa anche l'accesso all'omonima tabella del database.

Visto che oltre all'interfaccia 'dbdata' le slide dovevano offrire una loro specifica interfaccia aggiuntiva, si è introdotta una classe astratta intermedia 'slides' che eredita da 'dbdata', funge da base per 'slide' e 'slide_*' e richiede che venga implementato il metodo get_display().

Le slide finora implementate sono di quattro tipi, dettagliati di seguito.

3.5.1 Frequentazione aula

Preleva i dati dalle prenotazioni inserite in MRBS o in UniversityPlanner e li rende disponibili come slide.

3.5.2 Testo

Un 'semplice' spazio in cui scrivere.

Realizzare questo tipo di slide ha comportato la scrittura di un piccolo text formatter (per ora molto limitato, ma che potrà essere espanso se necessario) che attualmente permette di centrare, allineare a destra ed inserire tabulazioni, poi entrato a far parte della classe base per venire riutilizzato anche negli altri tipi di slide.

3.5.3 Grafica

Include semplicemente un file grafico inviato dall'utente e lo riscala, mantenendo l'aspect ratio, nella slide.

Molto comodo per presentare locandine di eventi.

3.5.4 Pannelli FV

Molto simile al tipo 'grafica', ma l'immagine viene generata automaticamente a partire dai dati forniti dall'inverter, raggiungibile solo tramite modem cellulare ed "interrogato" usando uno script lanciato da cron per limitare il traffico e velocizzare la visualizzazione.

Ha richiesto l'uso del parser XML e un po' di filtraggio di codice HTML.

3.5.5 Feed RSS

In fase di implementazione. Si presenta simile alla slide di testo, ma prende i dati da visualizzare da uno o più feed RSS/ATOM.

Richiede un parser XML, la riformattazione del testo (da HTML alla versione semplificata gestita dalle slide) ed una qualche forma di caching (il fetch dei feed può richiedere diversi secondi), analogamente ai dati dei pannelli FV.

Conclusioni

La prima cosa imparata è che, se si mostra al 'cliente' l'avanzamento del lavoro, le specifiche continuano a cambiare. Ciò può essere desiderabile solo se non si sono fissati termini di consegna e, soprattutto, se non si viene pagati 'a corpo'... Quindi a volte è meglio tenere la bocca chiusa su "quella funzioncina che potrebbe tornare utile", proponendola poi, solo dopo la consegna del resto, come possibile aggiunta.

Altra cosa importante: il 'cliente' ha *sempre* fretta. E molto raramente sa dire dall'inizio cosa gli serve. Quindi, tanti metodi di progettazione che porterebbero ad applicazioni belle, pulite ed ottimizzate non si riescono ad applicare nello sviluppo sul campo: la '*funzioncina che potrebbe tornare utile*' di cui sopra troppo spesso richiederebbe grosse riscritture, a meno di non averla "intuita" ed aver progettato di conseguenza.

In un progetto in continua evoluzione come questo, poi, spesso le specifiche finiscono per cambiare. Per es. all'ultima riunione è emerso che mantenere aggiornato lo stato delle carriere di Docenti e PTA, per l'Amministrazione, è troppo oneroso e considerato inutile. Come anche il modulo per aiutare nella suddivisione delle spese telefoniche (decisione politica: non si ripartiscono più a consuntivo) e quello per la gestione dei laboratori (essendo cambiata la responsabile sono cambiate anche le esigenze). E quindi tanto lavoro da

buttare[14].

Evoluzioni future includeranno probabilmente l'uso di SAML per l'autenticazione e l'autorizzazione (SSO d'Ateneo), e l'ottimizzazione del carico sul database con l'uso di memcached e 'piccole' modifiche all'astrazione del DB.

[14]Beh, meglio semplicemente disabilitare i relativi moduli, così che se torna a cambiare il vento siano già pronti!